IT507 Advanced Image Processing

ASSIGNMENT 1

DARSHIL - 202011034
VAIDIK -202011038

Q1: Objective:Up-sample and down-sample the image of Figure 1 by scale factor 4. Discuss the effect of changing sampling rate. Also observe the effect of different quantization levels (L=2, 4, 8, 16, 32, 64, 128, 256) for this image. (Figure 1 image : fig1.jpg)

Method and experiments :

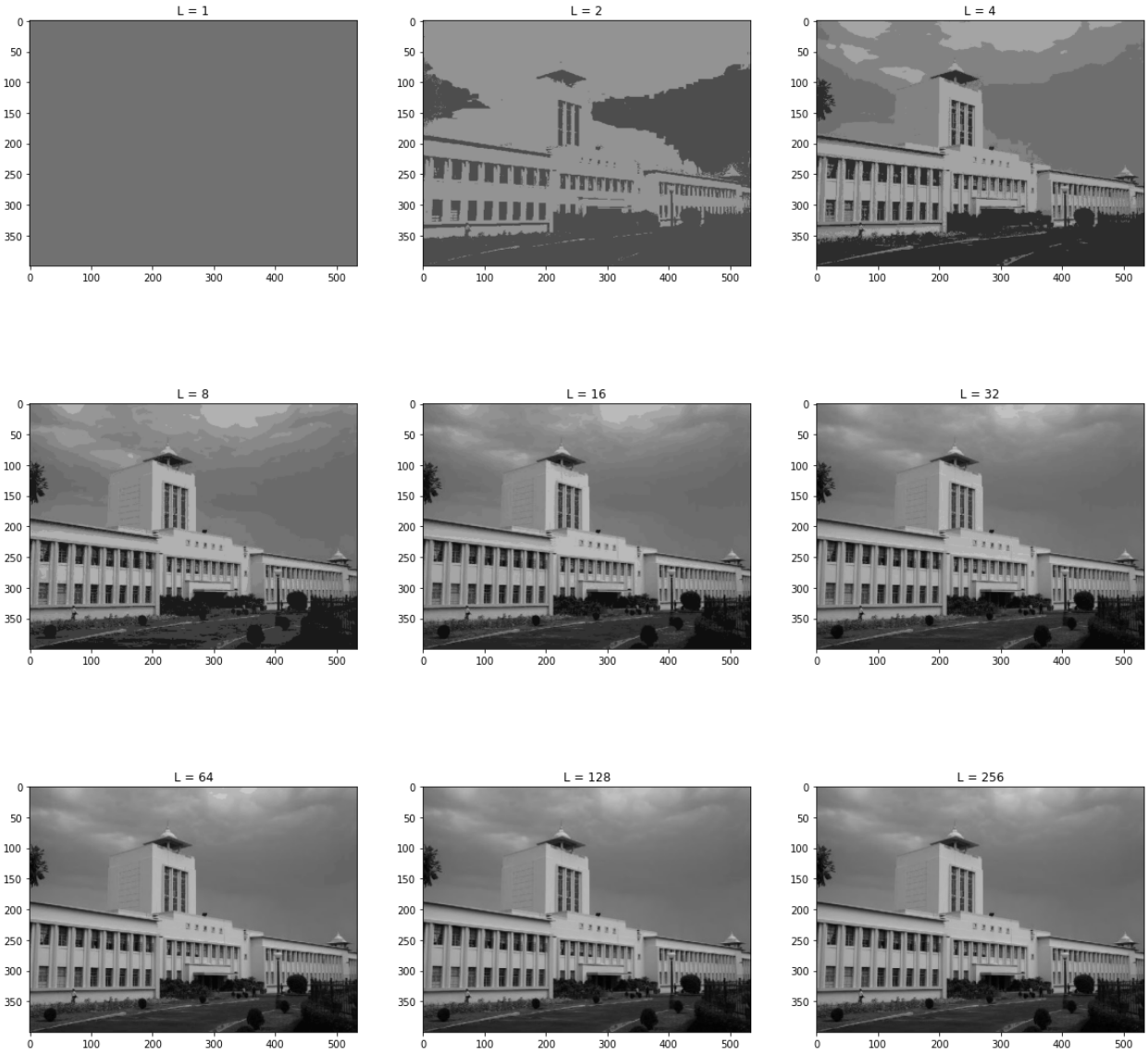
- First of all we read the image using the open CV library and convert it to grayscale.
- After preprocessing we move on to up sample and down sample the image by a factor of 4. To achieve this we scale the width and height of the image by a factor of 2 in the case of up sample and by a factor of ½ in case of downsampling. We have used cv2.INTER_AREA to do so.
- The way INTER_AREA works in downsampling is that it takes 4 adjacent pixels and calculates the mean of these 4 pixels and assigns the mean value to the pixel which represent these 4 pixel and then it moves to next set of 4 pixels.In case of upsampling it simply generates the copy of pixels and the number of copies depend on the scale of the factor. So, it changes the size of the image.
- To quantize the image we have PIL library which is the Python imaging library which provides python with image editing capabilities. We have used the .quantize() function of this library and this function takes in the level of quantization as argument and then quantizes the image based on the argument passed.

Conclusion:

- We concluded that upon downsampling the image the image became blurry and also the quality of the image decreases while upon upsampling no significant change is observed in the image. But the size of the image changes.
- Upon quantizing the image significant change is observed in the image between lower quantization levels i.e. 2,4,8,16,32,64 but no such change is observed at 128 and 256 levels.

(DOWN-SAMPLED IMAGE)

(UP-SAMPLED IMAGE)



(IMAGE AT DIFFERENT QUANTIZED LEVELS)

Q2.

Objective:Compute the mean and variance of Figure 2. Compute the same parameters for Figure1. Then add additive white Gaussian noise of mean 0 and standard deviation 20 to both the images and compute the means and variances. Explain your observations. Further, generate several such noisy images from Figure 1 and average all of them. Does the averaged image have less noisy artifacts? (Figure 2 image: baboon.jpg)

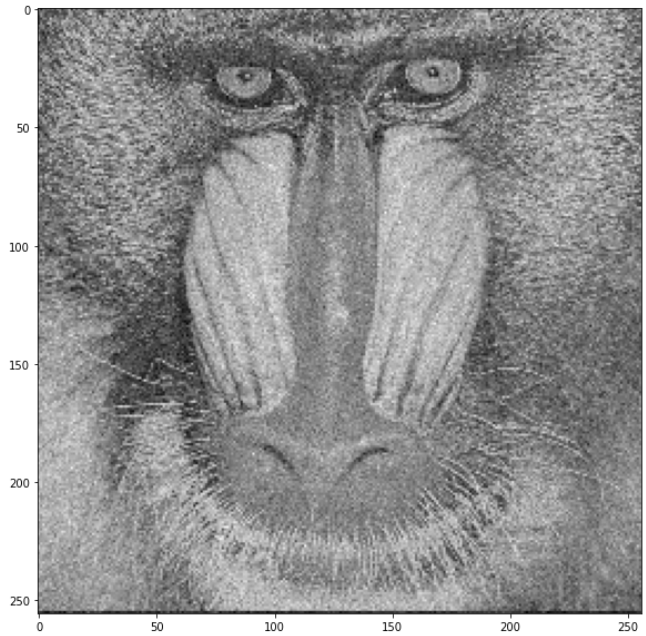
Method and experimentation:

- After the preprocessing we calculate the mean and variance of both the images.
- To calculate the mean and variance of the image we use Numpy library of Python and use the image as a simple array.
- We use np.mean() and np.var() functions to do so.
- Both of these functions take the image for which we want to calculate the mean and variance for as an argument.
- We use np.ramdom.normal() function of numpy to create white gaussian noise which takes the mean 0, variance 20 and the image dimension as arguments.
- We create 2 separate noises for the image as their dimensions are different.
- And in the final step we randomly generate 200 such noises and add them to our image and calculate its mean using the above method and simple array operations.

Conclusion:

- After adding gaussian noise the mean and the variance of the noisy image is different from the clear image. We see that upon adding white gaussian noise the image is visible to have become dotted because of the addition of noise.
- But to our amazement upon adding 200 random noise signals to the image and calculating mean image of all noisy images seems to be less noisy then the previous step.

(ADD GAUSSIAN NOISE TO THE IMAGE)



(ADD GAUSSIAN NOISE TO THE IMAGE)

(IMAGE AFTER ADDING 200 NOISE SAMPLES AND AVERAGING THEM)

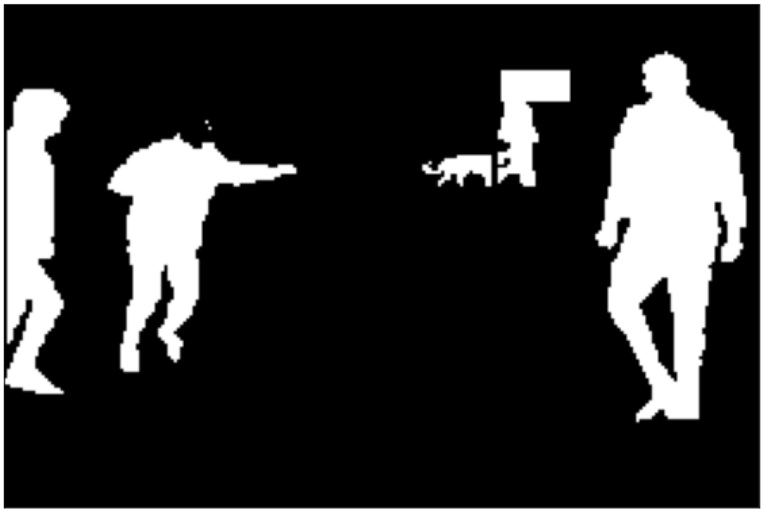
Q3.
Objective: Find out the number of objects from Figure 3. Label distinct objects with distinct colors.[Use the algorithm of finding out connected components.] (Figure 3 image: cc.jpg)

Method and Experiments:

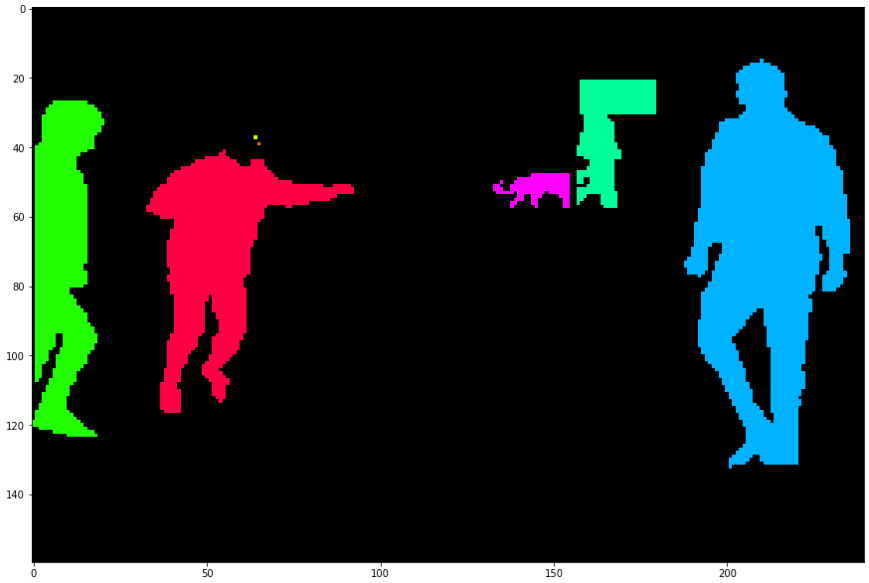
- As mentioned in the Objective we have used the connected components algorithm to find the distinct objects in the image.
- For this we used `cv2.connectedComponents()` function of open CV which takes the image as the argument.
- But before using the image as an argument in the above function we first turn the image into a binary image i.e. the background is black and foreground or the objects are white.
- For this we used the cv2.threshold() function. The first argument for this function is the source image which is a gray scale image.
- The second argument is the threshold which will classify the pixel values, the third argument is the max value assigned to pixels if the value of the pixel is more than the threshold and the fourth argument is the style of thresholding.
- We used the cv2.THRESH_BINARY.
- We pass this final binary image(image_bw) as an argument to cv2.connectedComponents() function.
- After which we color the distinct objects with distinct colors using the labels given to that object by cv2.connectedComponents() function.

Conclusion:

- The connected components algorithm works nicely if the image is two colored image or binary image.
- In this experiment the cap,ball and path are not calculated as the objects Because their colour is nearly equal to the background colour. So, while thresholding the image to convert two colored image these all things converted into black color.
- Another noticeable thing is that there are two white dots on the face of the person because of noise or light or something else. Connected components algorithm also considers these two dots as different objects.
- So, the connected component algorithm looks like a hard coded algorithm.



(FINAL IMAGE AFTER THRESHOLDING AND CHANGING IT TO BINARY)



(FINAL IMAGE AFTER COLORING THE DISTINCT OBJECTS)